Project 1

**Brandy Bridges**

February 8, 2010

SSE 554

Dr. Paul E. MacNeil

Contents

[Part I 3](#_Toc253402691)

[Introduction 3](#_Toc253402692)

[Web Application Security Fundamentals 3](#_Toc253402693)

[SQL Injection 5](#_Toc253402694)

[Defenses Against SQL Injection 6](#_Toc253402695)

[Validating User Input 6](#_Toc253402696)

[Using Parameterized Queries 7](#_Toc253402697)

[Using Stored Procedures 9](#_Toc253402698)

[Security Code Review 9](#_Toc253402699)

[Part II 13](#_Toc253402700)

[Creating a Wiki 13](#_Toc253402701)

# Part I

## Introduction

For the purpose of this project covering Secure Code, I used the MSDN article “Improving Web Application Security ( <http://msdn.microsoft.com/en-us/library/ms994921.aspx>) as my primary resource. This site is presented by Microsoft as a resource for developers who want to make their code more secure and less vulnerable.

The entire article is very comprehensive and for the purpose of this project I will narrow the scope to cover the following topics:

* Web Application Security Fundamentals
* Code Review

Software security as defined by Wikipedia is “Security, which encompasses all the measures taken throughout the application life-cycle to prevent exceptions in prevent exceptions in the security policy of an application or the underlying system (vulnerabilities) through flaws in the design, development, deployment, upgrade, or maintenance of an application.” On the site I used as a resource, security is defined as follows:

“Security is fundamentally about protecting assets. Assets may be tangible items, such as a Web page or your customer database — or they may be less tangible, such as your company's reputation.

Security is a path, not a destination. As you analyze your infrastructure and applications, you identify potential threats and understand that each threat presents a degree of risk. Security is about risk management and implementing effective countermeasures.”

We as developers must take this definition to heart and realize the security of our applications are as important at the beginning of the product life-cycle as it in the middle and the end. Just because we have finished our code, does not mean our job is complete. We must continually push the bounds of our code to ensure that it is secure always, and remember that we are ultimately responsible for any forgotten security measures.

## Web Application Security Fundamentals

Security leans on the following foundation elements:

* Authentication – poses the question: who are you? It is the process of identifying the users of applications and services. These “users” could be the actual users of the system, or perhaps other services, processes or computers who have “subscribed” to the system.
* Authorization – poses the question: what are you allowed to do? It is the process that defines the rules for resources and operations that an authenticated “user” allowed to access. Resources can include files, databases, tables, rows, and so on, along with other resources such as registry keys and configuration data.
* Auditing – is the key to non-repudiation, guaranteeing that a user cannot deny performing a task or transaction.
* Confidentiality – also known as *privacy* is the process of making sure that data remains private and confidential. Encryption is most frequently used to enforce confidentiality.
* Integrity – the guarantee that data is protected from change, whether it is accidental or deliberate. Integrity is a huge concern, especially for data passed across networks. Integrity for data while in transit is usually accomplished by using hashing techniques and message authentication codes.
* Availability – means systems remain available for authenticated users. Many hackers are trying to succeed with denial of service attacks to crash an application or to make so overwhelmed that others cannot access the system.

The best way to start building secure Web applications is to be familiar with the possible threats to those applications. After coming up with a comprehensive list of all possible threats, you can start to design the application with those threats as ever-present reminders of how to implement your security. Five of the most common threats you should have on your list are:

* SQL Injection
* Cross-Site Scripting Attacks
* Denial of Service Attacks
* Buffer Overflows
* Session Hijacking

Let’s look at least one of these threats, SQL Injection, because it is my favorite to play around with and test sites to see if they have implemented simple security measures.

## SQL Injection

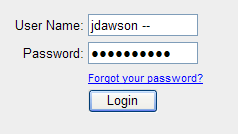
Wikipedia defines SQL Injection as “a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed.” In a nutshell, just try typing a literal string that is recognized as a valid SQL statement and see what happens. Let’s look at the username and password combination from an application I work on in my job. Instead of typing in a random password to try to get in the system, I will type ‘ or 1=1 for the password. I am using one of those combinations betting that there is some SQL in the code for the page that will use the password as part of the conditional when validating the user.



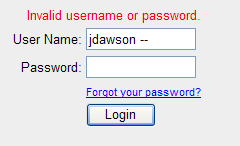
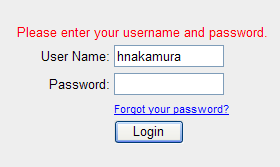
The result for this web application is:



This tells me that at least the login fields are not vulnerable to this particular attack. I did not get any unexpected behavior from the system and it did not return any errors from the database to reveal any information about the database structure. Now let’s try a different attack. I will append the user name with the characters that indicate a database comment (‘--‘) for an Oracle database. By adding the “comment” characters, I could effectively get rid of any conditionals that existed in and SQL in the code of the page that occurred after the username.



Because I am a confident developer, I expect the following error message, , because I already know that there is no user that matches the input “jdawson --". Let’s see what happened.



Thankfully, I got an error message back that indicates that the combination is invalid. The error message did not tell me whether the username or password was invalid, just than one of those inputs was invalid, so I cannot tell if I can use that username for further attacks.

I used a username that I know is correct in the first example, but if I were trying to manipulate vulnerabilities, I would probably pick a username like “Admin” or “Administrator” because many systems use those names for global administrators, or I could possibly use “sa” and the default SQL Server “sa” password and hedge my bets that the people who built the system are letting SQL Server handle user authentication and did not change the “sa” password.

I work with Oracle Databases, and I like [SecurityFocus](http://www.securityfocus.com/).com for a look at how SQL Injection is handled with Oracle databases.

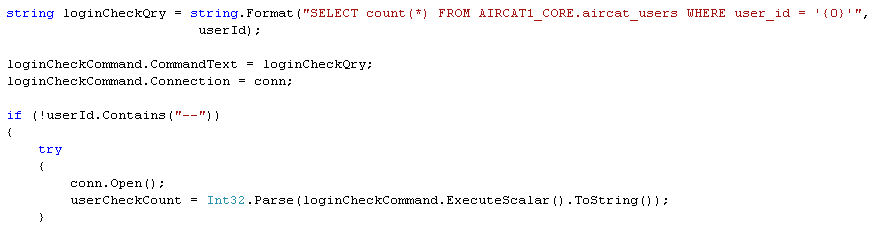
## Defenses Against SQL Injection

There are several ways to defend your application against SQL Injection attacks, including Validating User Input, Using Parameterized Queries, and Using Stored Procedures.

### Validating User Input

SQL Injection requires the use of special characters and/or special sequences of characters in order to be successful. We as developers have to be ever vigilant in our quest to secure our applications and make sure we are not only validating our user input to make sure it is valid as data to be captured and stored, but we must also validate that the inputs are not dangerous to our applications. The special characters/sequence of characters we have to validate are:

* A single quote (‘) – we should replace a single quote mark with 2 single quotes before sending the data to the database for evaluation.
* Two hyphens – Two hyphens will comment out anything that comes after the hyphens, all inputs with two hyphens should be returned as invalid. We can accomplish this with a simple if statement.



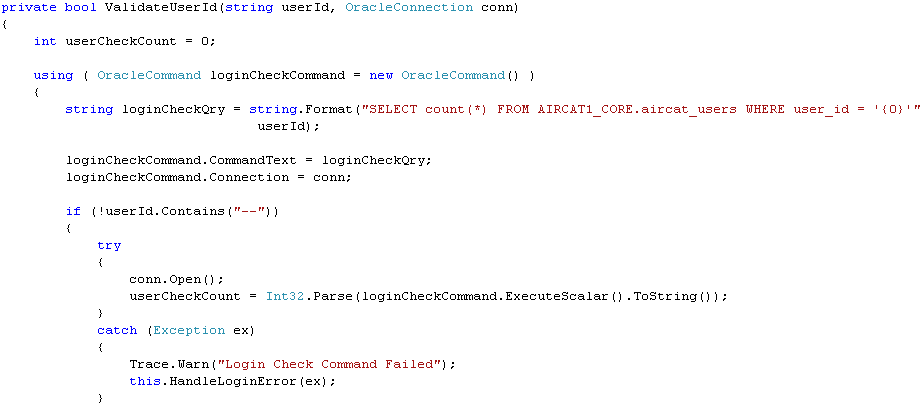
What this code means is: if the userId does not contain those characters, then proceed with the login code. I used a simple “If” statement to guard against some dangerous SQL Injection.

* System Table Names – This is for all Relational Database Management Systems, so all developers should be familiar with the system tables for the RDBMS they are working with so that we can guard against the system tables being compromised. System Tables contain system level data for the actual database, like table names, column names, constraint names, objects in the database, etc. If we allow these tables to be accessed then our database could be easily corrupted.

### Using Parameterized Queries

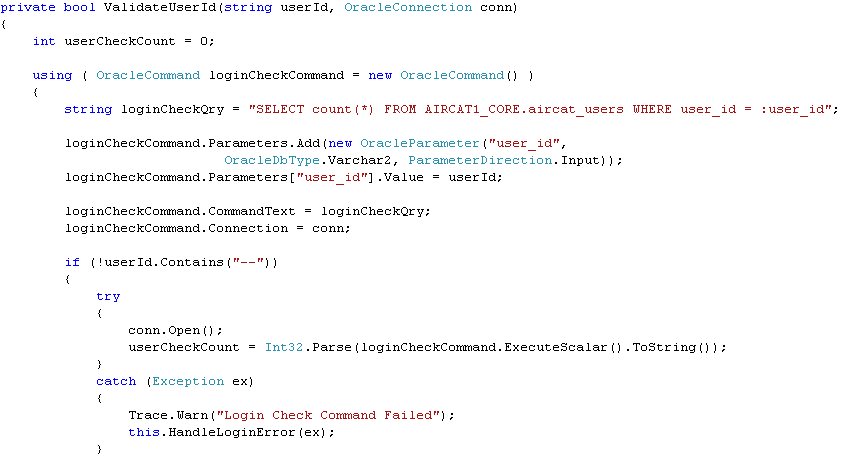
MSDN defines a parameterized query as “A query that contains placeholders for values to be bound at query execution.” What this means is that instead of building dynamic queries with all the quote marks involved and formatting the input to concatenate with the query string, we can just provide some actual parameters to the query and let the database handle the rest. There are a few good reasons for using parameterized queries but the best reason by far is to prevent SQL Injection. Consider the following login example:

This code shows the query to check is a user exists without a parameterized query.



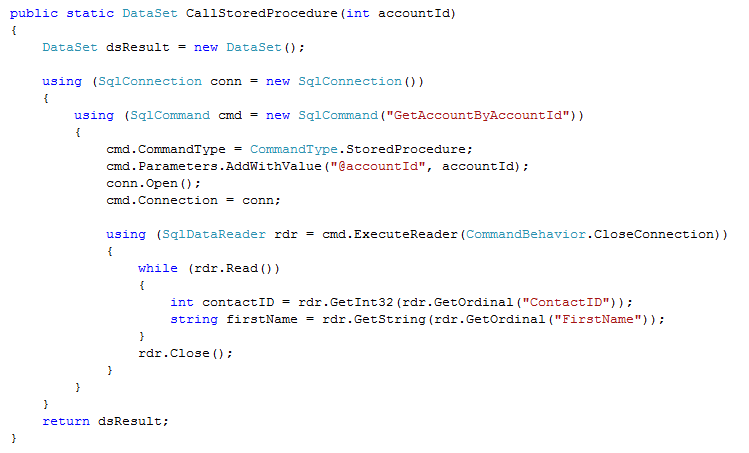
The string variable is build by simply concatenating strings using the string.Format method. There are no actual parameters for the query, because there are no parameters defined for the database call itself.

This code however, does use parameters to bind values before executing the database call.



### Using Stored Procedures

Using stored procedures (and functions when applicable) to execute database queries and transactions add an additional layer or abstraction to a software system. It provides “separation of duties” between the code and the database as well as gives the system an added layer or protection against SQL Injection. By restricting access to the data to stored procedures only, we don’t have to create and database users with excessive permissions just to ensure table level access. The following is an example of making a stored procedure call, with parameters added (remember the previous topic…).



This is a basic example using SQL Server as the database, but you can make the same calls using an Oracle Database. Using stored procedures offer the same type of protection as parameterized queries. You have more control over the data being passed and manipulated.

## Security Code Review

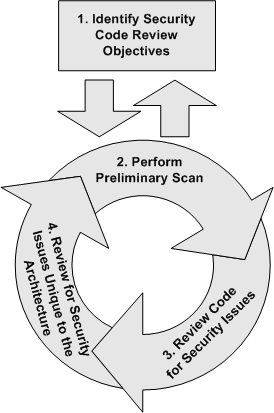
A thorough code review can reveal security shortcomings in the application before the software is ever tested, much less deployed. The best way to start a comprehensive code review is to have a checklist and a good idea of the weaknesses you are trying to find in the code. I used the “Secure Code Review Tutorial” is from [MSDN](http://msdn.microsoft.com/en-us/library/ms998364.aspx) as a resource for this section.

In order to complete a Security Code Review, you should at bare minimum have the source code available and in addition having the following is also helpful:

* Architecture or component diagrams
* Data Flows
* Data Schemes

The output of the code review will be a list of identified security vulnerabilities that will have to be prioritized for repair before a release or deployment.

The steps for a Security Review are as follows (diagram from MSDN):



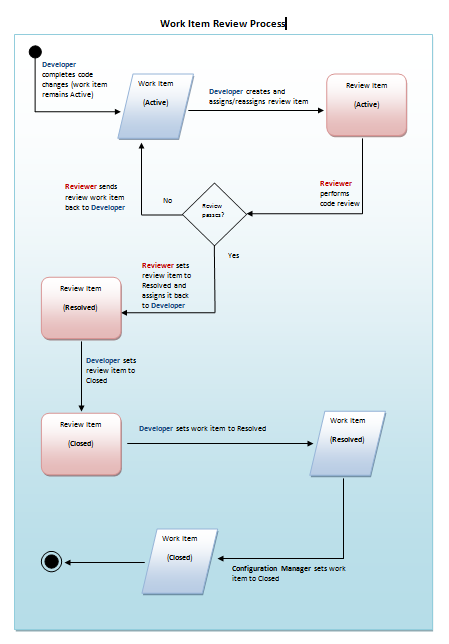
You have to begin a review with a working knowledge of what standards and flaws you are looking for in the source code. It is best to go into the code review with a list of possible security issues to be on the lookout for. Some of those issues are:

* SQL injection
* Cross-site scripting
* Input/data validation
* Authentication
* Authorization
* Sensitive data
* Code access security
* Exception management
* Data access
* Cryptography
* Unsafe and unmanaged code use
* Configuration
* Threading
* Undocumented public interfaces

An effective review will have goals and constraints for time, types of issues to be discussed, and out of scope items (issues that will not be discussed). You should also determine the objectives of the code review. In order to determine the objectives, the following questions should be considered:

* What common coding errors apply to the source code being reviewed?
* What is the scope of the review?
* If using a threat model, which of the threats applies to the code being reviewed?
* Example Security Objectives
  + Make sure that all untrusted input to the component is passed to a validation routine before it is used.
  + Check error handling to make sure that exceptions are caught consistently and caught close to their source.
  + Check calculations whose results are used for memory allocation or buffer access for numeric overflow or underflow.
  + Check cryptographic routines to make sure secrets are cleared quickly.

For most groups, short reviews covering small pieces of code is much more manageable than trying to review. At my place of employment, we conduct code reviews at the time of check-in and no code is integrated into a release until it passes the review. The following is a diagram of the workflow we use (which I helped to develop and implement):



A little explanation of the previous figure is warranted: At my job we use Team Foundation Server as our configuration management system, and we call System Change Requests (SCRs) Work Items because that is what TFS calls them. So the steps for working a Work Item is

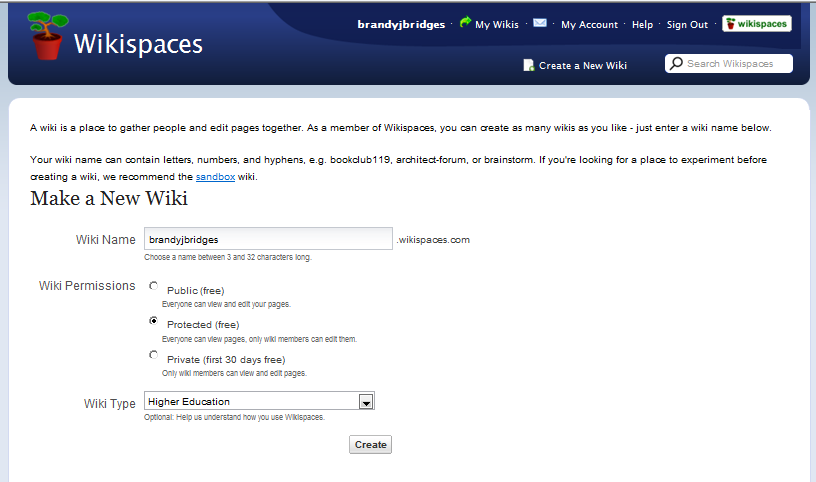
* 1. Developer is assigned a work item; the status of the work item is set to Active.
  2. Developer makes necessary changes (source code changes, database changes, etc) and checks the changes in, associating those changes with the assigned work item.
  3. Developer creates a new review work item related to the original work item and assigns it to a Reviewer.
  4. Reviewer reviews the changes, tests functionality of changes.
  5. Decision Tree:
     + If changes are approved
       - The review work item has its status changed to Resolved and is assigned back to the original developer
     + If changes are not approved the review item status stays active and is assigned back to original developer with any notes and comments about why it did not pass the review process.
     + Then the cycle repeats until the changes are approved.
  6. Work Items passing the review:
     + Review item status is set to Closed.
     + Original Work Item status is set to Resolved and ready for release.

I feel like this is a very good way to start implementing Security Code Reviews for any team of developers. It is very clear and concise. We are left with no doubts as to what issues we should be on the lookout for when reviewing each other’s code. I would attempt to try to become familiar enough with these practices to make them a part of my peer code review process. This could streamline the security code review process and add an extra look at the security of the application.

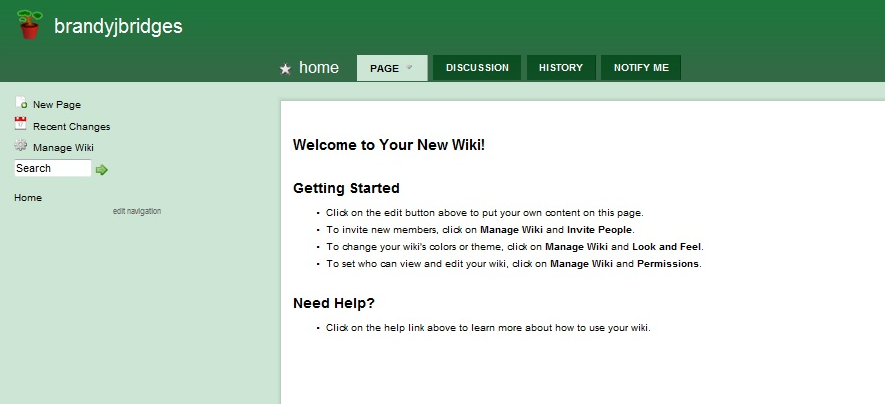
# Part II

## Creating a Wiki

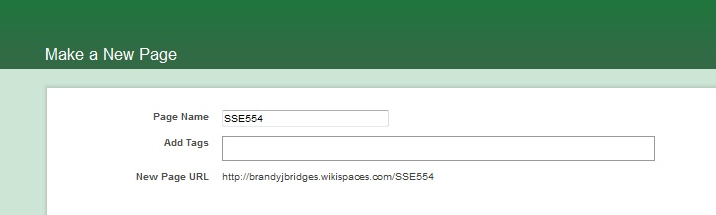
For this project I have created new wiki at <http://brandyjbridges.wikispaces.com/>, which will also be used for other classes this semester. It was very easy to create, I just registered as a new user and hit the ground running. The first thing I did was create a new wiki by clicking the “Create a New Wiki” link in completing the page that looks like this:



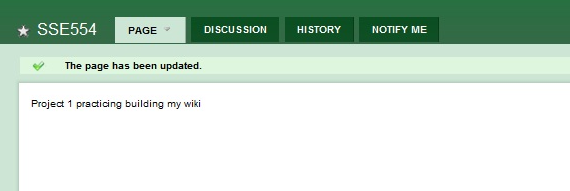
Then once I had my new wiki, I read the getting started page:



The next step was easy, I just started to add pages. I gave a little thought to the navigation and decided that I needed a page for each class I am currently taking, also a page for each project for each class. This is how I added my first page:

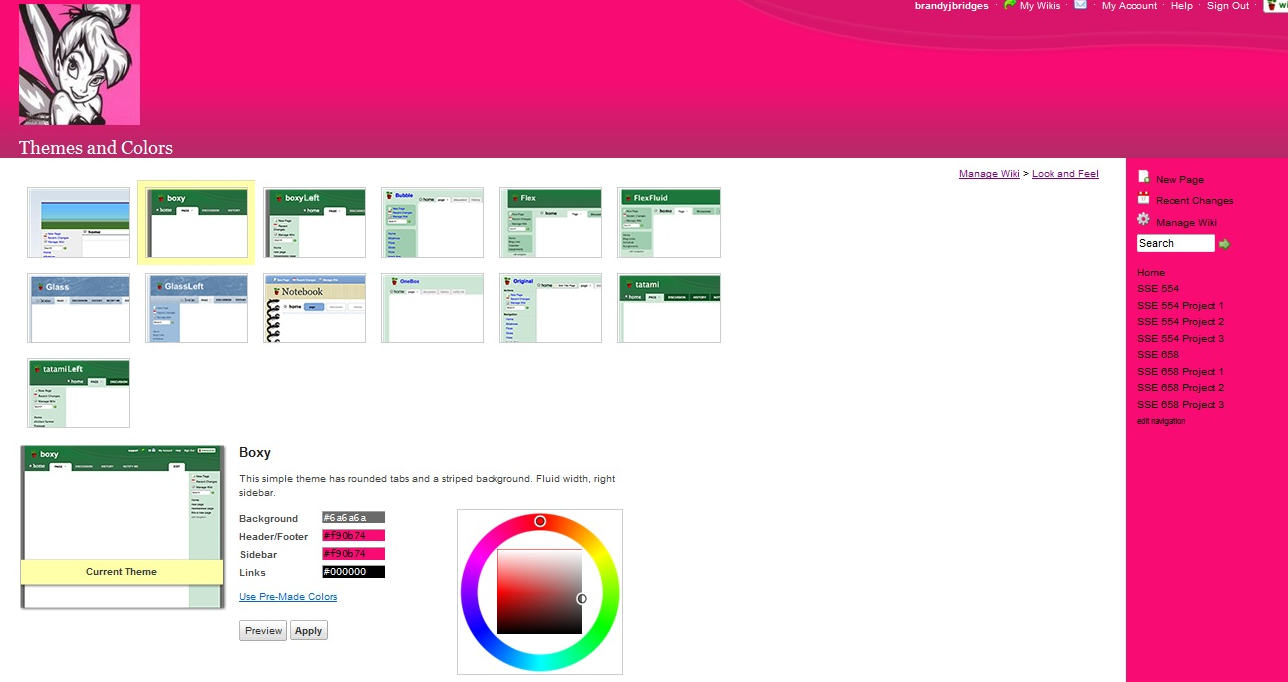


Then I added a little content as kind of a placeholder until I could come back and fill it in a little better:



Once I built the pages I wanted, I started playing with the settings to make it more my own.

* I changed the layout from the default layout.
* I changed the icon of the wiki (I like Tinkerbelle).
* I also played with the colors so that it would match my new icon a little better



I decided to upload my project to the site as well…just to practice, so this project is also available on my wiki.